(12) UK Patent Application (19) GB (11) 2 259 590 (13) A

(43) Date of A publication 17.03.1993

(21) Application No 9217274.1

(22) Date of filing 14.08.1992

(30) Priority data
(31) 9119700 (32) 14.09.1991 (33) GB

(71) Applicant
International Computers Limited

(Incorporated in the United Kingdom)

ICL House, 1 High Street, Putney, London, SW15 1SW, United Kingdom

(72) Inventor
James Press

(74) Agent and/or Address for Service
D C Guyatt
International Computers Limited, Patents & Licensing, Six Hills House, London Road, Stevenage, Herts., SG1 1YB, United Kingdom

(51) INT CL^5
G06F 9/44

(52) UK CL (Edition L)
G4A APL APX

(56) Documents cited
EP 0458495 A2

(58) Field of search
UK CL (Edition K) G4A AAP APX
INT CL^5 G06F 9/44
On-line database: WPI

(54) **Modifying a cryptographic facility of a data processing system**

(57) A cryptographic facility uses object-oriented techniques to define a hierarchical structure representing cryptographic services. Whenever an instance of an object class is created, Fig 5, or destroyed, Fig 6, the instance is automatically respectively registered with a list of services supported by the system or de-registered from the list. This facilitates modification of the system, e.g. by adding a new service.

The cryptographic facility is used in communications between applications (14, Fig 1).

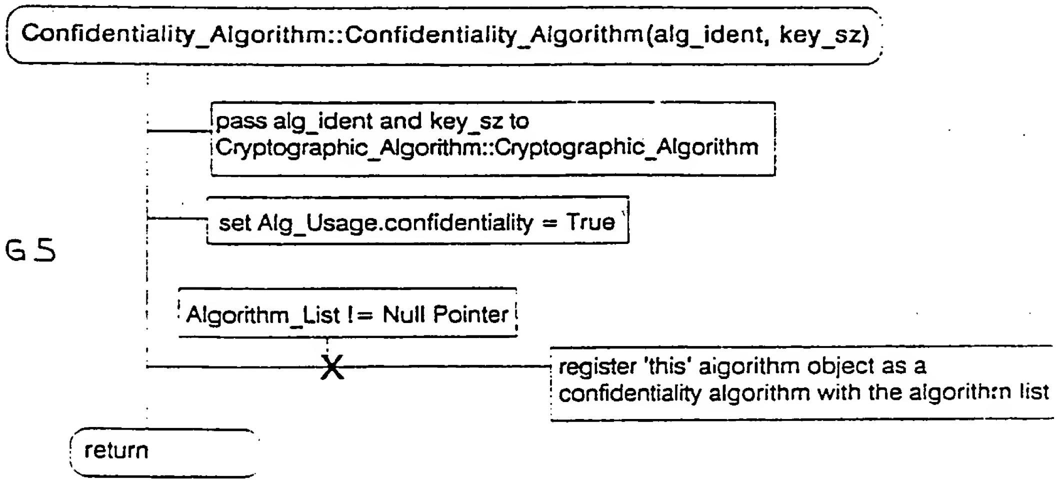
Confidentiality_Algorithm::Confidentiality_Algorithm(alg_ident, key_sz)
pass alg_ident and key_sz to Cryptographic_Algorithm::Cryptographic_Algorithm
set Alg_Usage.confidentiality = True
Algorithm_List != Null Pointer
register 'this' algorithm object as a confidentiality algorithm with the algorithm list
return

FIG 5

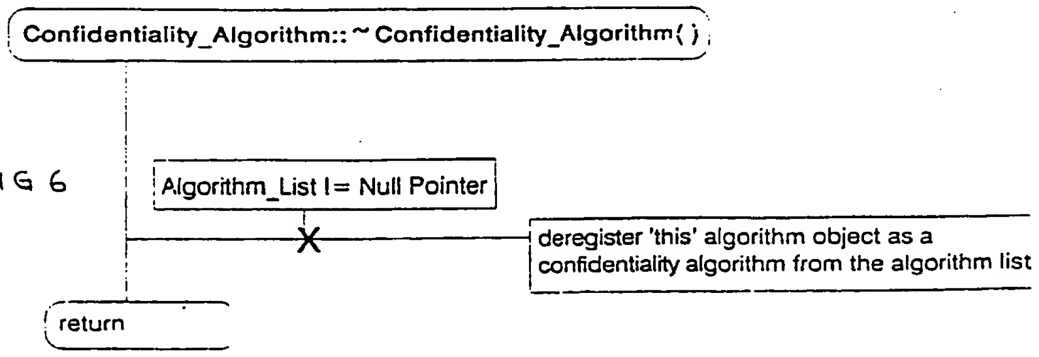
Confidentiality_Algorithm::~Confidentiality_Algorithm( )
Algorithm_List != Null Pointer
deregister 'this' algorithm object as a confidentiality algorithm from the algorithm list
return

FIG 6

GB 2 259 590 A

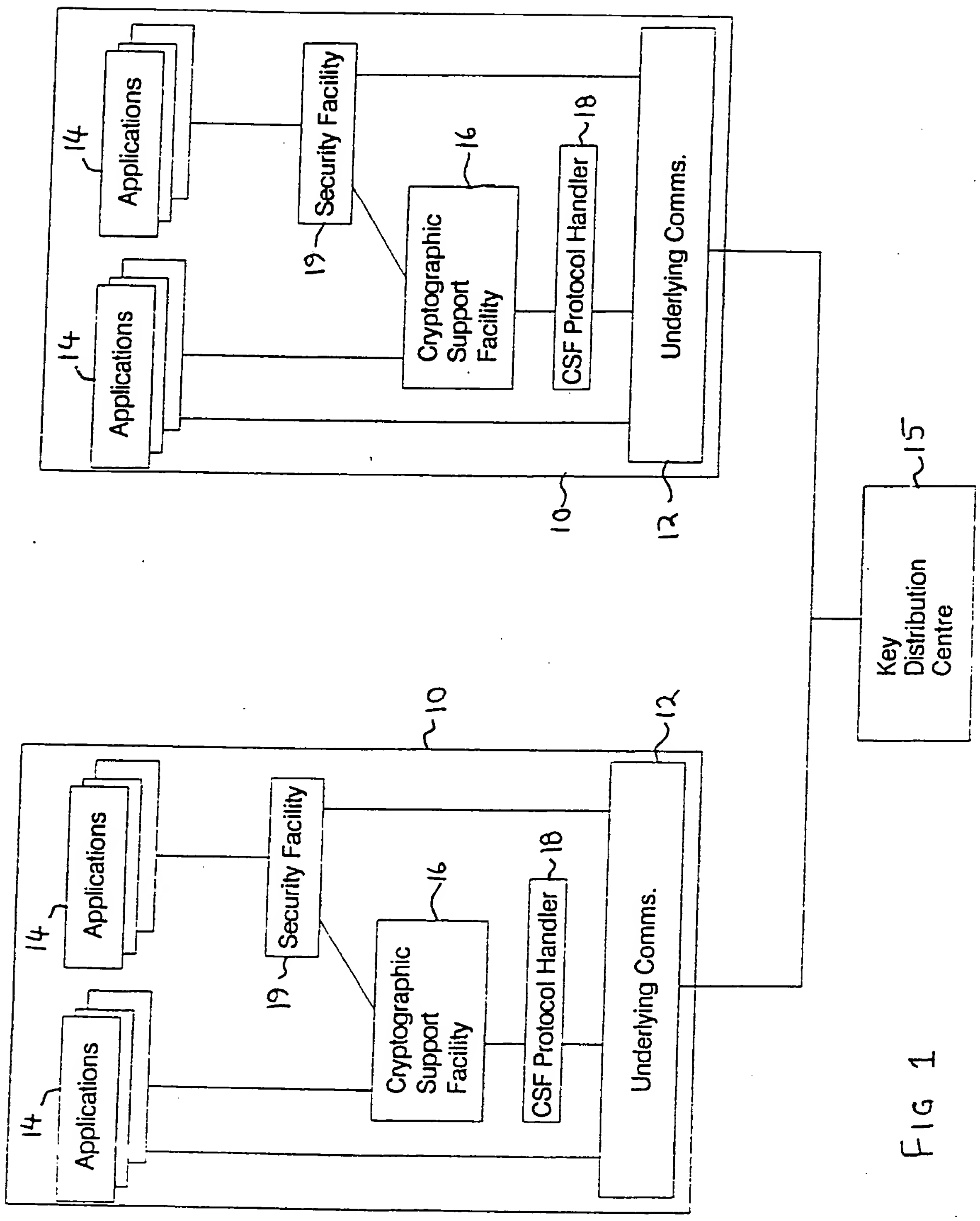
14
Applications
14
Applications
Security Facility
19
Cryptographic Support Facility
16
CSF Protocol Handler
18
Underlying Comms.
10
12
Key Distribution Centre
15
14
Applications
14
Applications
Security Facility
19
Cryptographic Support Facility
16
CSF Protocol Handler
18
Underlying Comms.
10
12

FIG 1

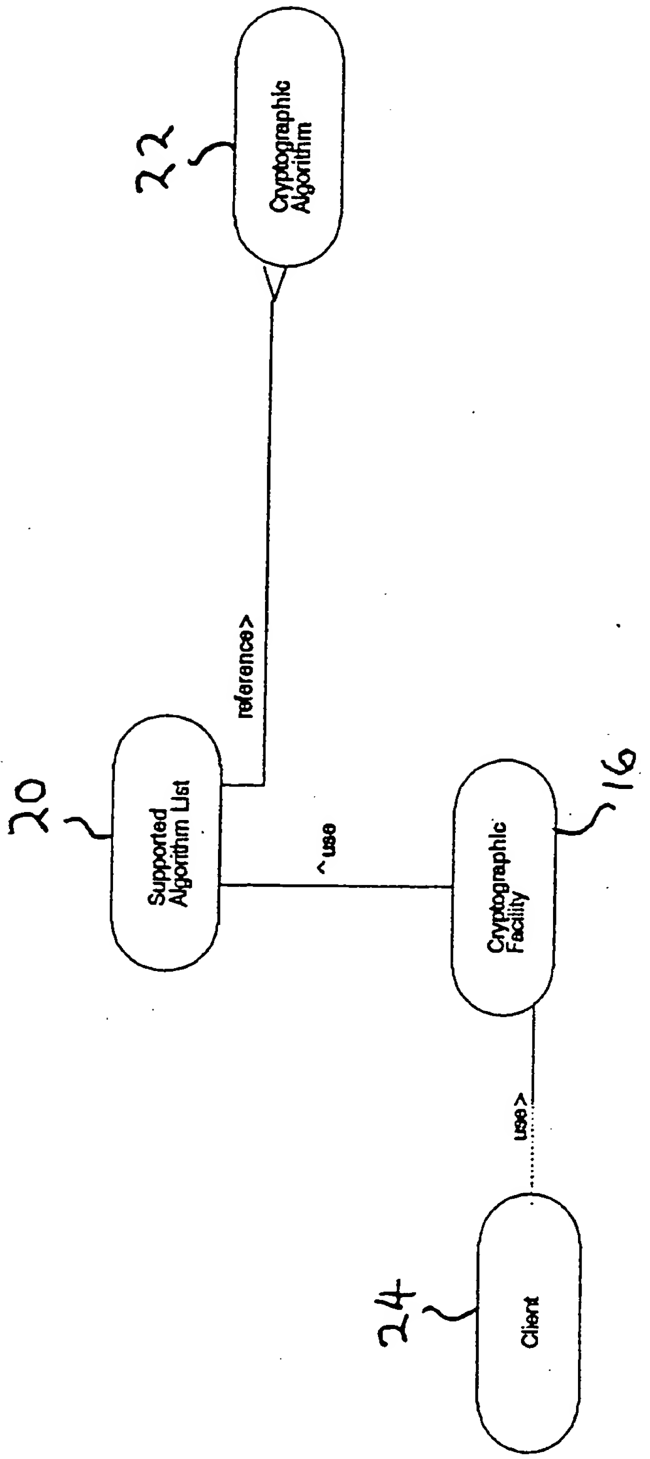
22
Cryptographic Algorithm
reference>
20
Supported Algorithm List
^use
Cryptographic Facility
16
use>
24
Client

FIG 2

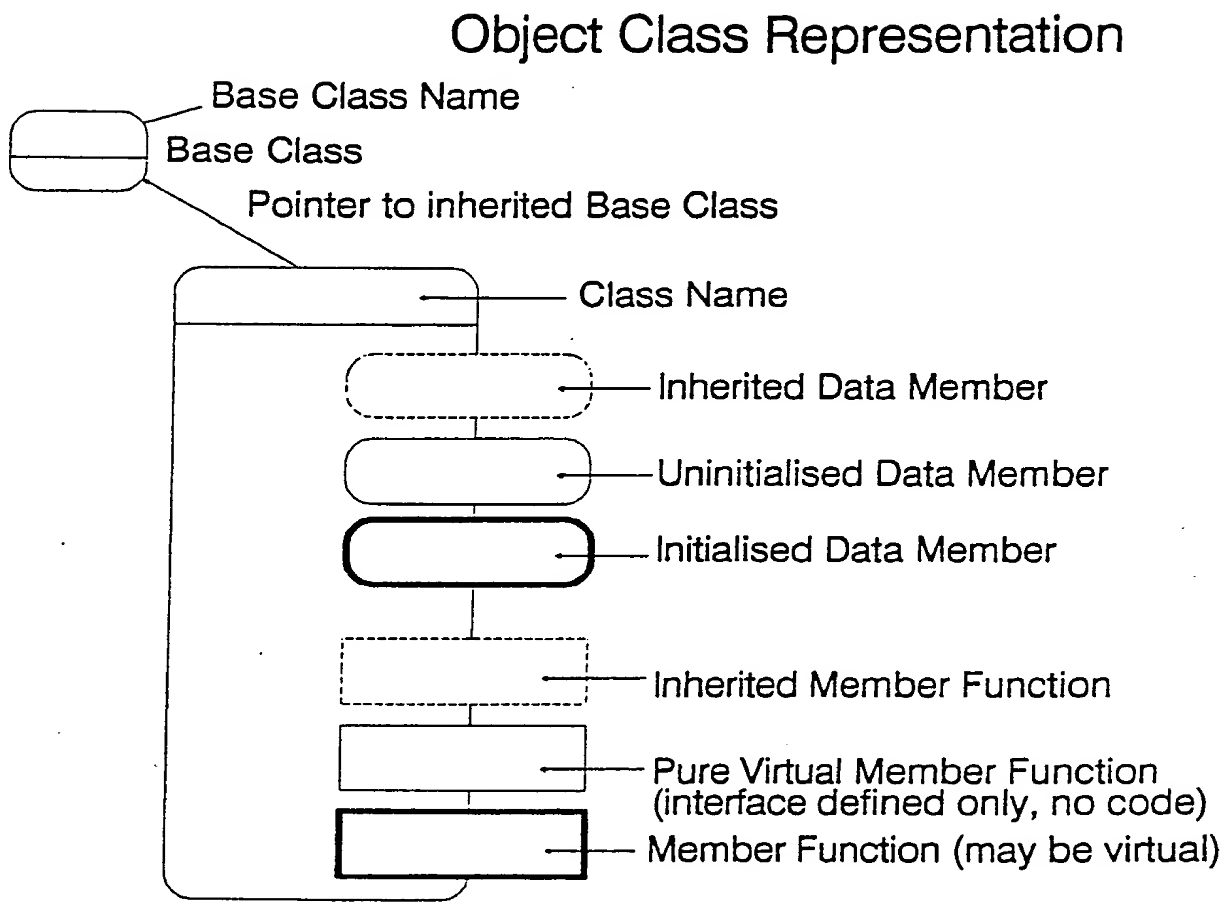
Object Class Representation
Base Class Name
Base Class
Pointer to inherited Base Class
Class Name
Inherited Data Member
Uninitialised Data Member
Initialised Data Member
Inherited Member Function
Pure Virtual Member Function (interface defined only, no code)
Member Function (may be virtual)

FIG 3

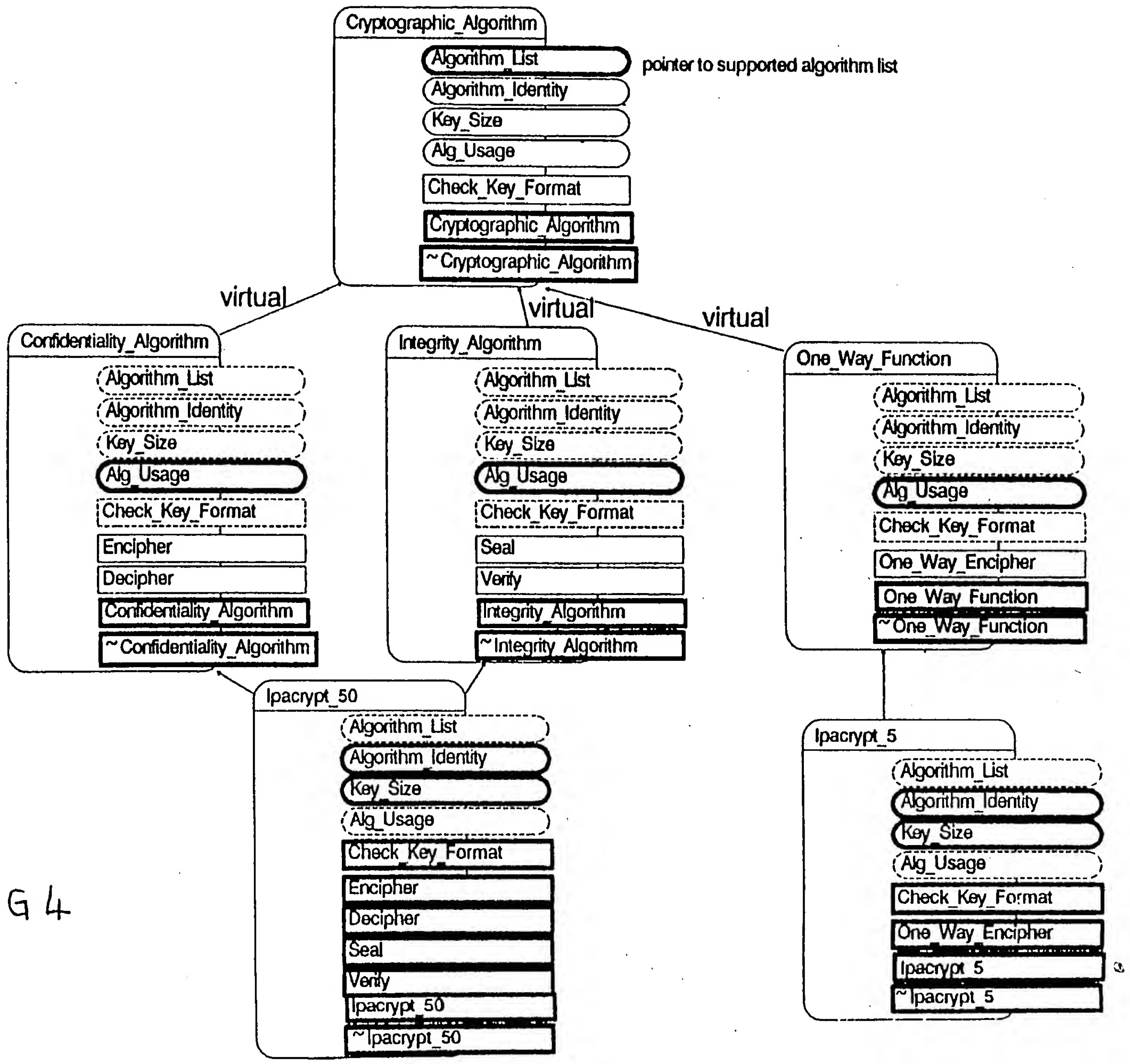
Cryptographic_Algorithm
Algorithm_List
pointer to supported algorithm list
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
Cryptographic_Algorithm
~Cryptographic_Algorithm
virtual
virtual
virtual
Confidentiality_Algorithm
Algorithm_List
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
Encipher
Decipher
Confidentiality_Algorithm
~Confidentiality_Algorithm
Integrity_Algorithm
Algorithm_List
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
Seal
Verify
Integrity_Algorithm
~Integrity_Algorithm
One_Way_Function
Algorithm_List
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
One_Way_Encipher
One_Way_Function
~One_Way_Function
Ipacrypt_50
Algorithm_List
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
Encipher
Decipher
Seal
Verify
Ipacrypt_50
~Ipacrypt_50
Ipacrypt_5
Algorithm_List
Algorithm_Identity
Key_Size
Alg_Usage
Check_Key_Format
One_Way_Encipher
Ipacrypt_5
~Ipacrypt_5

FIG 4

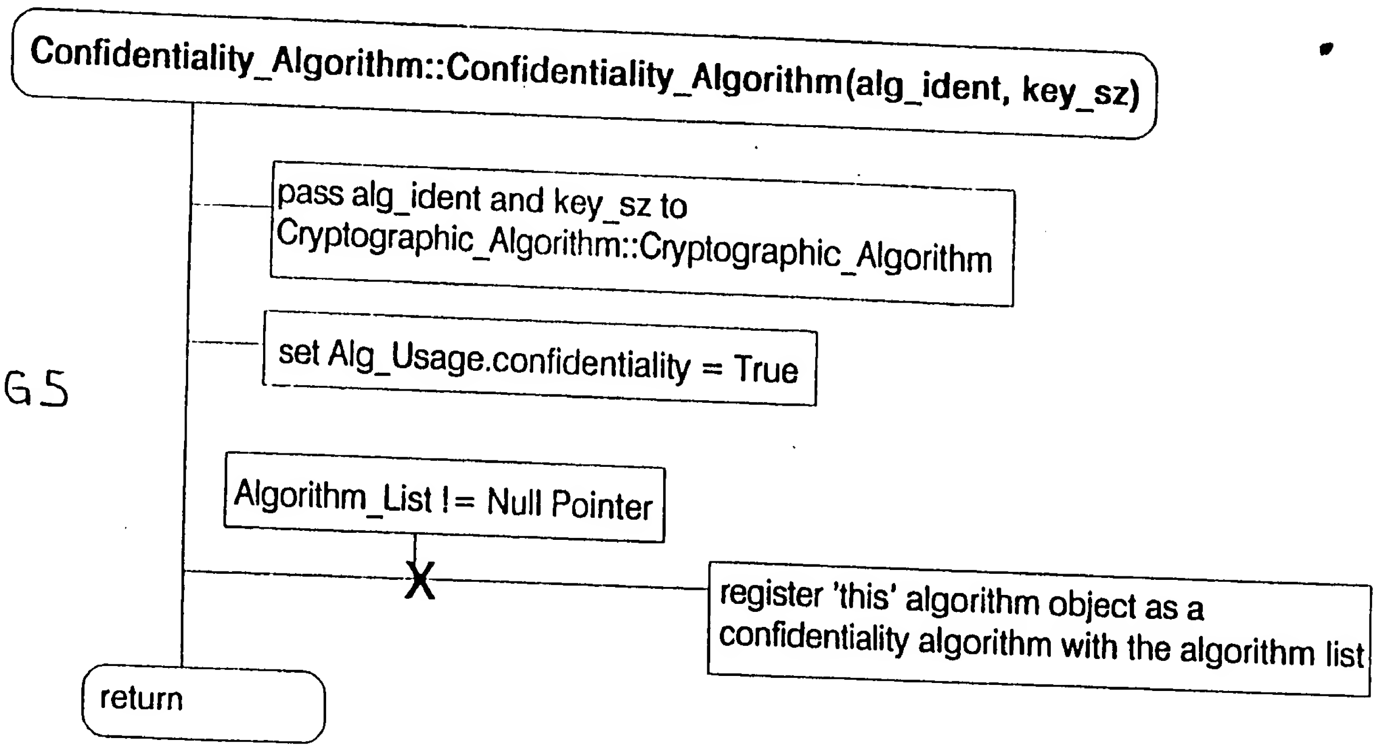
Confidentiality_Algorithm::Confidentiality_Algorithm(alg_ident, key_sz)
pass alg_ident and key_sz to Cryptographic_Algorithm::Cryptographic_Algorithm
set Alg_Usage.confidentiality = True
Algorithm_List != Null Pointer
register 'this' algorithm object as a confidentiality algorithm with the algorithm list
return

FIG 5

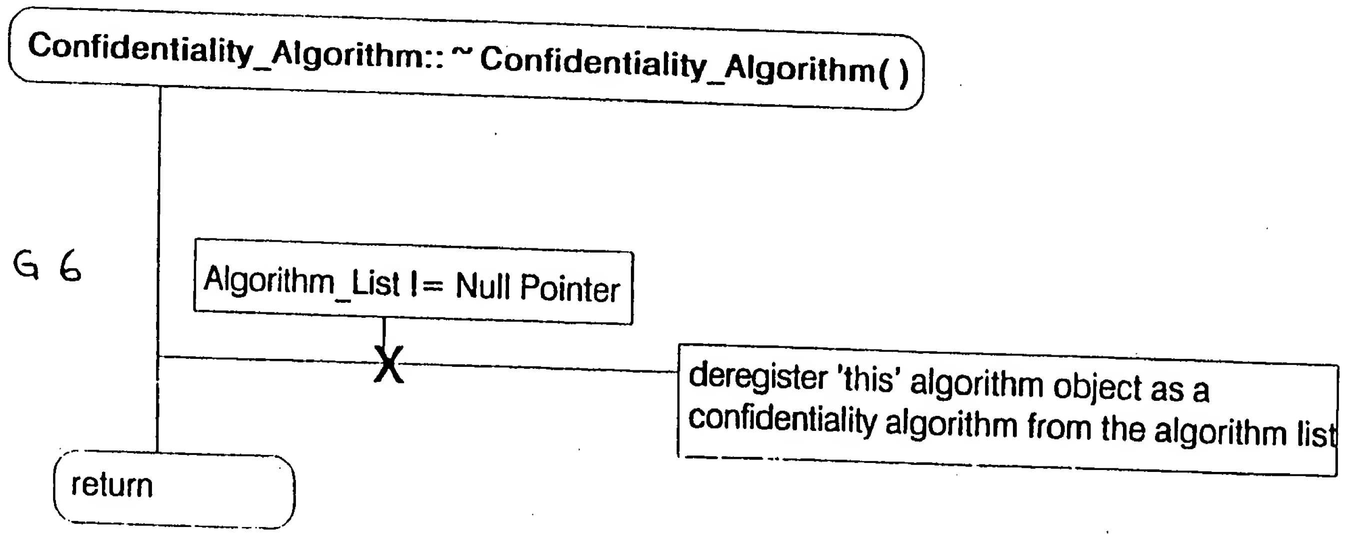
Confidentiality_Algorithm:: ~ Confidentiality_Algorithm( )
Algorithm_List != Null Pointer
deregister 'this' algorithm object as a confidentiality algorithm from the algorithm list
return

FIG 6

CRYPTOGRAPHIC FACILITY FOR A DATA PROCESSING SYSTEM

Background of the invention

This invention relates to a cryptographic facility for use in a data processing system.

Typically, such a cryptographic facility may be required to provide a number of different services such as data encryption, key generation, and so on. Each service may have several different categories: for example, various types of cryptographic algorithm may be required.

A list of available services may be stored in a table. For example, a list of available cryptographic algorithms may be held in a table referred to herein as a "supported algorithm list".

In existing systems of this type, the introduction of a new service (e.g. a new cryptographic algorithm) presents considerable difficulties. For example, in the case of a new algorithm, it is necessary not only to introduce the new code for the algorithm, but also to modify the supported algorithm list to include a reference to the new algorithm. This in general involves a large amount of recompilation.

The object of the present invention is to provide a way of simplifying the introduction of new services in such a system.

Summary to the invention

According to the invention there is provided a cryptographic facility for a data processing system, comprising:

- means for defining a hierarchical structure of object classes defining cryptographic services, wherein each class can inherit functions and data from a base class in the hierarchy,

- means for creating an instance of an object class, defining a cryptographic service, and for automatically registering that instance with a list of services supported by the system,

- means for destroying an instance of an object class, defining a cryptographic service, and for automatically de-registering that instance with the list of services supported by the system, and

- means for accessing the functions of an instance of an object class, defining a cryptographic service, through a base class of the object class in the hierarchy of object classes.

It will be seen that the problem of creating new services is facilitated by the hierarchical structure of the classes. Also, when a new instance of a class is created, it is automatically registered in the services list, so that it is not necessary for this to be done manually.

Brief description of the drawings

Figure 1 is a block diagram of a data processing system including a cryptographic facility in accordance with the invention.

Figure 2 is a block diagram showing the objects which interact with the cryptographic facility.

Figure 3 is a schematic diagram illustrating notation used to describe object classes.

Figure 4 is a schematic drawing showing object classes relating to cryptographic algorithms.

Figures 5 and 6 are flow charts showing constructor and destructor functions for a confidentiality algorithm object class.

Description of an embodiment of the invention

One cryptographic facility for a data processing system will now be described by way of example with reference to the accompanying drawings.

Referring to Figure 1, the system comprises a number of data processing units 10. The processing units can communicate with each other by way of communications facilities 12 in each processing unit.

Each processing unit includes a number of applications 14 (i.e. applications programs). The applications can communicate with each other by way of the communications facility 12.

Each processing unit also includes a cryptographic support facility (CSF) 16 which provides cryptographic services to the applications. The CSF can communicate, by way of a CSF protocol handler 18, with a key distribution centre 15, to obtain cryptographic keys. A security facility 19, is also provided to restrict access to some of the applications.

The present system is implemented by means of a technique known as object oriented design, using the C++ programming language. For further details of object oriented design and C++, reference is made to "C++ Primer", S.B. Lippman, Addison Wesley (1989).

Conventional design methods are based on the use of functional decomposition, where large problems are broken into smaller ones by concentrating on dividing the major steps which appear in the flow of control. The main limitation of designs produced by such methods is that they are not particularly good at coping with change, which often accounts for the majority of a software system's life.

The main distinction between Object-Oriented Design (OOD) and conventional design is that it is object-oriented and to process-oriented.

Central to OOD is the concept of an 'object'. An object embodies an abstraction of information which is meaningful to its clients. It has the following properties:

a) An object has state encompassing all of the (usually static) properties and the current (usually dynamic) values of each of these properties.

b) An object has behaviour defined by the 'services' it provides to its clients (other objects or programs).

The terms 'operator' or method' are sometimes used instead of 'service'. Clients do not generally directly access the data in an object, they send 'messages' to the object requesting services to be carried out to access or manipulate the actual data in the object. The terms 'request' is sometimes used instead of 'message'. A service may be classified as:

Modifier	- alters the state of an object;
Selector	- accesses the state of an object without modifying it;
Iterator	- permits the parts of an object to be visited;
Constructor	- creates an object initialising its state;
Destructor	- destroys the object freeing its state.

In this way, objects provide information-hiding by data encapsulation. By avoiding direct client access to the data, one can guarantee certain integrity constraints of the object. It is also possible to change the object implementation without affecting the clients, unless there is a change in the nature of the services provided.

c) An object has identity and is denoted by a name. To make a request, a client identifies the object which is to perform the service and names the request. Requests may take arguments (including references to other objects) and the service may return one or more results.

d) An object is an instance of some class. A class contains a common structure and a common behaviour applicable to all instances of the class. Classes can be derived from other classes (inheritance).

Referring now to Figure 2, this is a schematic diagram showing the CSF and the other objects within the system, with which the CSF interacts.

The CSF uses a supported algorithm list 20. This contains a set of pointers to all the cryptographic algorithms 22 that are currently available to the CSF.

The CSF is used by a user 24, which is typically one of the applications programs 14 (Figure 1). When the user requires to use the cryptographic facility, (e.g. to encrypt an item of data), it sends a message to the CSF, requesting it to perform the desired cryptographic operation.

It can be seen that the CSF thus provides a simple interface for cryptographic operations, such that the user does not need to be concerned with details of the algorithm, the choice of key, etc.

Each of the objects shown in Figure 2 is implemented by creating a specific instance of a particular object class. For example, a cryptographic algorithm is implemented by creating an instance of an object called "cryptographic -algorithm".

These cryptographic algorithm object classes are arranged in a hierarchical structure, as will now be described.

Referring to Figure 3, this shows the notation used in this specification to describe the object classes and their hierarchical structure.

As shown in Figure 3, each object class has a class name, and consists of a number of data members and member functions (using C++ terminology). Each class may have a pointer to a base class, ie. to its parent class within the hierarchical structure.

Data members that are inherited from the base class are denoted by broken lines, uninitialised data members are denoted by continuous lines, and initialised data members are denoted by bold continuous lines. Similarly, inherited member functions are denoted by broken lines, pure virtual member functions (see below) by continuous lines, and member functions in general by bold continuous lines.

A pure virtual function is a function for which the interface is defined in a base class and its implementation is provided by a derived class.

Two special operations supported by all classes are the 'constructor' used to create and initialise an instantiation of the class, and the 'destructor' used to delete an instantiation of the class.

In C++, a constructor and destructor are automatically generated by the compiler. The writer of a class may provide code for a constructor which is to be executed once the compiler-generated code has created the object and initialised the data members and the base classes of the object (base classes are initialised before derived classes). The writer may also provide initialisation lists which are used to initialise data members and base classes. In C++, a constructor takes the name of the class, for example (using the notation <class name>:: <function name> to represent the name of a function) Shift_Register_Generator::Shift_Register_Generator is a class constructor.

A class may also contain code for a destructor which is executed before compiler-generated code has deleted the object. In C++, a destructor is the name of the class prefixed by a '~', for example Shift_Register_Generator_::~Shift_Register_Generator is a class destructor.

Referring now to Figure 4, this shows how, in this particular example, cryptographic algorithms are implemented using a hierarchical structure of object classes.

The highest level of the hierarchy is an object class called "cryptographic algorithm". This class contains a number of data members and member functions as shown.

The function "cryptographic-algorithm" is a constructor function which is used to create new instances of the class. Similarly, the function "~ cryptographic-algorithm" is a destructor function which is used to destroy instances of the class that are no longer required.

The next level of the hierarchy consists of a number of sub-classes, representing the main categories of cryptographic algorithm provided by the CSF: confidentiality algorithms, integrity algorithms, and one-way functions. (Other sub-classes may also be provided) These categories provide generic interfaces for functions common to the category.

Object classes for two specific algorithms ("Ipacrypt 5" and "Ipacrypt 50") are derived from the classes shown in Figure 4. It is possible for object classes to have multiple inheritance characteristics: for example, in this case, Ipacrypt 50 is derived from two of the main sub-classes.

Referring now to Figure 5, this is a flow chart showing the constructor function for the confidentiality-algorithm object class (Fig. 4).

As shown the first action of this constructor function is to pass the parameters of the function to the constructor function of the base class ("cryptographic-algorithm"). This will then, in turn, initialise the values for the various members that are inherited from the base class.

The constructor function then sets the data member that indicates that the usage of the algorithm in question is for confidentiality.

Finally, the constructor function tests whether the inherited pointer to the supported algorithm list object is null. If it is non-null, the algorithm in question is registered with the supported algorithm list as being derived from the confidentiality algorithm sub-class.

Referring to Figure 6, this is a flow chart, showing the destructor function for the confidentiality - algorithm object class. It can be seen that this automatically de-registers the algorithm.

The constructor and destructor functions for the other main sub-classes act in a similar manner.

As shown in figure 5, Ipacrypt 50 is derived from the classes confidentiality algorithm and integrity algorithm. Thus when an object of this algorithm is created, the constructor of the confidentiality algorithm class will register this object with the supported algorithm list as a confidentiality algorithm, and the constructor of the integrity algorithm class will also register this object with the supported algorithm list as an integrity algorithm.

Note that the supported algorithm list only needs to know the identity of the algorithm being registered and from which of the main sub-classes it is derived from, it need have no knowledge about the actual class of the algorithm, The CSF will call upon the supported algorithm list providing an algorithm identity and requesting a pointer to one of the main sub-classes of an object of that algorithm. For example, the CSF could request a pointer to the confidentiality algorithm class component of an Ipacrypt 50 object. The CSF can then invoke those functions of the object which are inherited from the sub-class (e.g. 'encipher').

Installing a new algorithm involves linking in the module containing the class of the new algorithm which is derived from one or more of the main sub-classes, and creating an object of this class by invoking its constructor. The algorithm will then automatically register itself with the supported algorithm list for use by the CSF.

CLAIMS

1. A cryptographic facility for a data processing system, comprising:

- means for defining a hierarchical structure of object classes defining cryptographic services, wherein each class can inherit functions and data from a base class in the hierarchy,

- means for creating an instance of an object class, defining a cryptographic service, and for automatically registering that instance with a list of services supported by the system,

- means for destroying an instance of an object class, defining a cryptographic service, and for automatically de-registering that instance with the list of services supported by the system, and

- means for accessing the functions of an instance of an object class, defining a cryptographic service, through a base class of the object class in the hierarchy of object classes.

2. A cryptographic facility for a data processing system substantially as hereinbefore described with reference to the accompanying drawings.

Patents Act 1977
Examiner's report to the Comptroller under Section 17 (The Search Report)

Application number
GB 9217274.1

Relevant Technical fields

(i) UK Cl (Edition K) G4A (AAP,APX)

(ii) Int Cl (Edition 5) G06F 9/44

Search Examiner
B G WESTERN

Databases (see over)
(i) UK Patent Office

(ii) ONLINE DATABASE: WPI

Date of Search
6 OCTOBER 1992

Documents considered relevant following a search in respect of claims 1-2

Category (see over)	Identity of document and relevant passages	Relevant to claim(s)
A	EP 0458495 A2 (TEXAS INSTRUMENTS)	

Category	Identity of document and relevant passages	Relevant to claim(s,

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family, corresponding document.

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).

Published 1993 at The Patent Office, Concept House, Cardiff Road, Newport, Gwent NP9 1RH. Further copies may be obtained from